Electrical & Computer Engineering
University of Waterloo

ECE 419 Project
Topic #1

Report

Michael Anthony Soares

Due date: July 20th, 2011
Submission date: July 6th, 2011

Structure

The client-server pair of programs for the DSS mini-certificate generator and verifier are split up into four different classes, explained in more detail later on in this report:

- Client – handles the client part of the program pair
- Server – handles the server part of the program pair and calls functions on a DSSSig object related to generating and verifying certificates and other data
- DSSSig – handles all computations involved in generating and verifying certificates, as well as any other computations to check individual values, system parameters, and key pairs
- Cert – instantiated as an object to store certificate-related data

All large integers are manipulated using the *BigInteger* library. All client-server communication is done through the *Socket* and *SocketServer* libraries.

All compilation and running instructions can be found in *README.md*.

System Parameters & Key Pair

The system parameters and the key pair used in the assignment are:

- p =
  168199388701209853920129085113302407023173962717160229197318545484823101018386724351964316301278642143567435810448472465887143222934545154943005714265124445244247988777471773193847131514083030740407543233616696550197643519458134465700691569680905568000063025830089599260400096259430726498683087138415465107499
- q = 959452661475451209325433595634941112150003865821
- g =
  943891927763273985898453269803498145264338690934127823454309460592065688040051816008558259061429672718725483758777389498758125404332234449684613507894613850437750299639006381231834351335372621529733554984329953645051389125697558596236498663751353531793626707987717707118474306269548642698889883711135675028
  52
- CA Private Key =
  432398415306986194693973996870836079581453988813

- CA Public Key = 49336018324808093534733548840411752485726058527829630668967480568854756416567496216294919051 91014868618662270686970232166446509470324736864650682101529030248099045013028061692922691724 62551470632923017242976806834012586361821855991241311700775484507542940837288850755169851449 449849200101384928972720692571 60

Client
The client is programmed to run continuously until it is told to forcefully shut down by the server through user input. On connection to the server, it generates a random public key and sends it to the server. Once sent and server acknowledges receipt, the client simply waits for a message, waits for user input, and sends the user input to the server; this continues in a loop until it is told to stop by the server. Once stopped by the server, all connections are closed.

Server
The server, like the client, is run in a loop until the user chooses to exit. The server waits for a connection from a client. It initially receives the public key from the client and then asks the client/user if they wish to generate a certificate, verify one, or exit.

If the user chooses to generate a certificate, the server prompts the user for their identity, verifies it, then prompts the user controlling the server-side for an expiry date. If the expiry date is in the correct format, the user is asked to continue and then the certificate is generated by passing the needed parameters to the *generateCert()* function in the *DSSSig* object; a *Cert* object is then passed back to the server, formatted, and sent to the client.

If the user chooses to verify a certificate, the server prompts the user for each line of the certificate individually (i.e., the message, *r*, and *s*, respectively). Once these parameters are received, a *Cert* object is created and verified through the *verifyCert()* function in the *DSSSig* object.

If the user enters any other input, the server forcefully shuts off the client.

DSSSig
The *DSSSig* class is made up of several functions. *generateCert()* is simply an intermediate function that takes in the required parameters *id* and *pkClient* and passes them to the private function, *computeRS()*. *computeRS()* generates the required parameters, *r* and *s*, based on a random *k* value. It also takes the message passed in and runs the *hashMsg()* function to *SHA1* hash the message. If the *k* value is not in the correct range, all three values are recalculated until three valid values are found. Once these values are acquired, a *Cert* object is created and returned; this represents the actual certificate, which can be later formatted however one desires.

verifyCert() takes in a *Cert* object with the certificate information to verify. It first verifies that the *r* and *s* values are in the correct range. If they are not, the function returns *false*. If they are, the computations for *w*, *u1*, *u2*, and *v* are done. *v* and *r* are then compared. If they are equal, the certificate is valid and the function returns *true*. If they are not, the certificate is not valid and the function returns *false*.

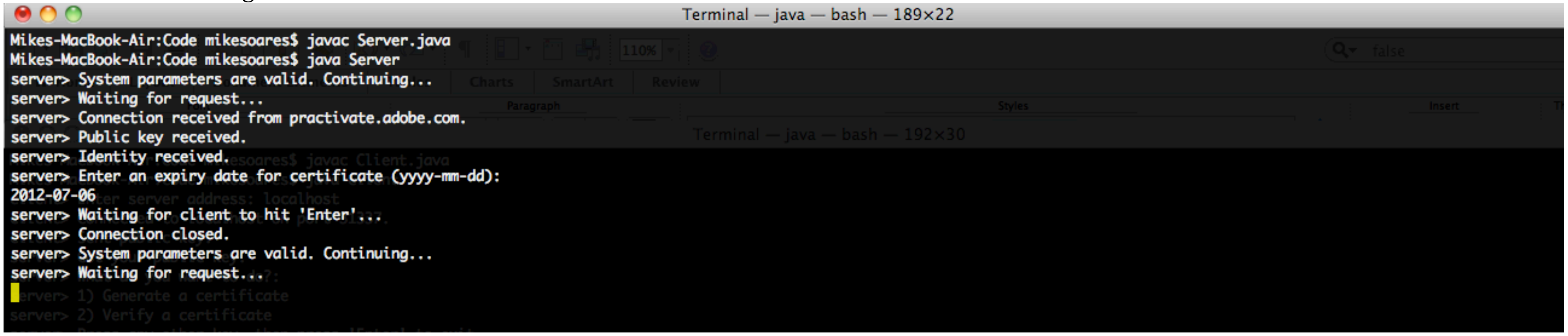
There are several functions that check if the *p*, *q*, *g*, and keypair values are valid. These are *checkP()*, *checkQ()*,*checkG()*, and *checkKeypair()*, respectively. These are called by *checkValues()* when verification is needed.

hashMsg() simply takes any message (a string) passed as a parameter and returns the *SHA1* hash as a string.

Cert
Cert is simply made up of several getters and setters for the message, *r*, and *s* values. The constructor takes in these values as parameters in the same order as a string, and two *BigIntegers*, respectively.

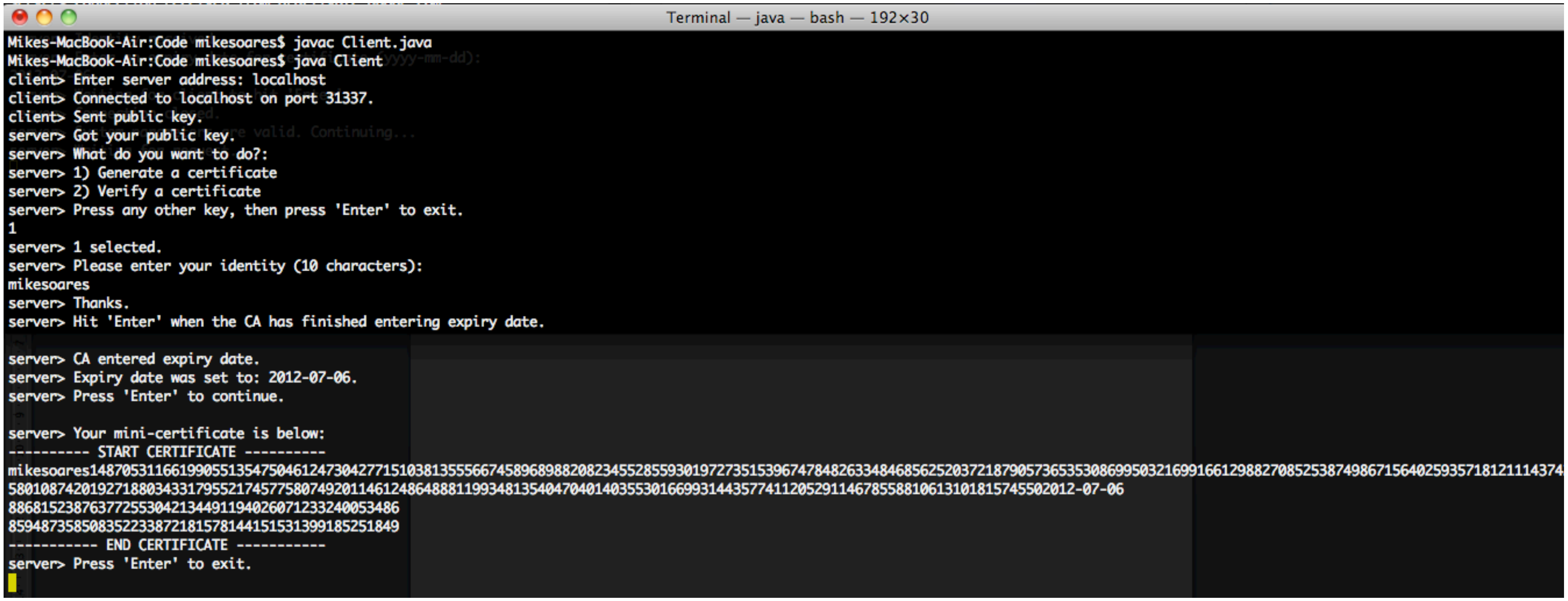
Sample Output

Generating a certificate – server-side:



```
Mikes-MacBook-Air:Code mikesoares$ javac Server.java
Mikes-MacBook-Air:Code mikesoares$ java Server
server> System parameters are valid. Continuing...
server> Waiting for request...
server> Connection received from practivate.adobe.com.
server> Public key received.
server> Identity received.
server> Enter an expiry date for certificate (yyyy-mm-dd):
2012-07-06
server> Waiting for client to hit 'Enter'...
server> Connection closed.
server> System parameters are valid. Continuing...
server> Waiting for request...
```

Generating a certificate – client-side:

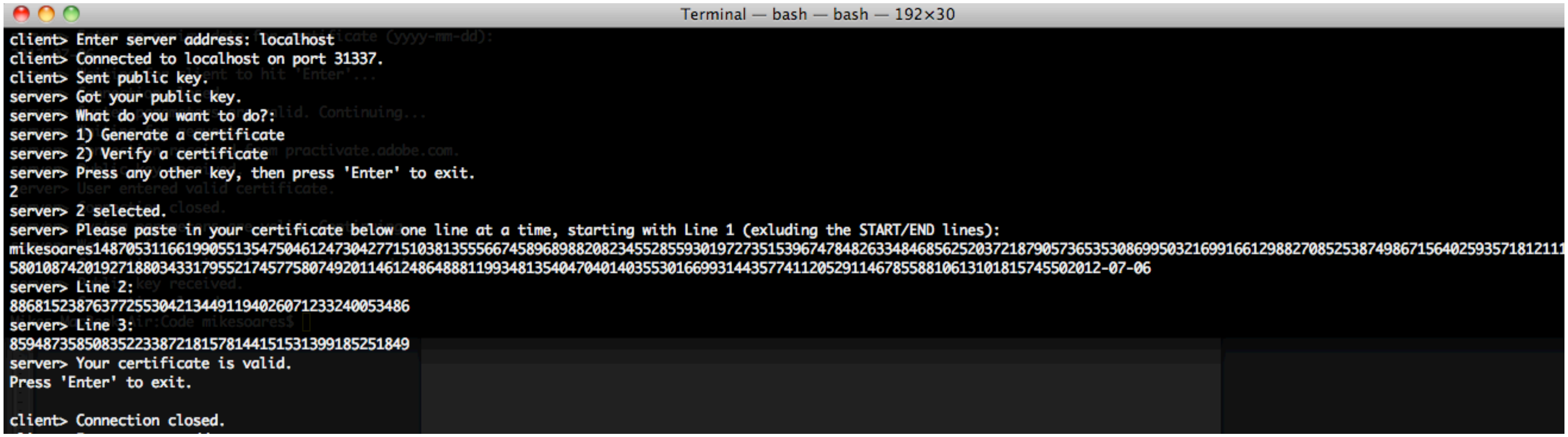


```
Mikes-MacBook-Air:Code mikesoares$ javac Client.java
Mikes-MacBook-Air:Code mikesoares$ java Client
client> Enter server address: localhost
client> Connected to localhost on port 31337.
client> Sent public key.
server> Got your public key.
server> What do you want to do?:
server> 1) Generate a certificate
server> 2) Verify a certificate
server> Press any other key, then press 'Enter' to exit.
1
server> 1 selected.
server> Please enter your identity (10 characters):
mikesoares
server> Thanks.
server> Hit 'Enter' when the CA has finished entering expiry date.

server> CA entered expiry date.
server> Expiry date was set to: 2012-07-06.
server> Press 'Enter' to continue.

server> Your mini-certificate is below:
---------- START CERTIFICATE ----------
mikesoares1487053116619905513547504612473042771510381355566745896898820823455285593019727351539674784826334846856252037218790573653530869950321699166129882708525387498671564025935718121114374
58010874201927188034331795521745775807492011461248648881199348135404704014035530166993144357741120529114678558810613101815745502012-07-06
886815238763772553042134491194026071233240053486
859487358508352233872181578144151531399185251849
----------- END CERTIFICATE -----------
server> Press 'Enter' to exit.
```

Verifying a certificate – client-side:



```
client> Enter server address: localhost
client> Connected to localhost on port 31337.
client> Sent public key.
server> Got your public key.
server> What do you want to do?:
server> 1) Generate a certificate
server> 2) Verify a certificate
server> Press any other key, then press 'Enter' to exit.
2
server> 2 selected.
server> Please paste in your certificate below one line at a time, starting with Line 1 (exluding the START/END lines):
mikesoares14870531166199055135475046124730427715103813555667458968988208234552855930197273515396747848263348468562520372187905736535308699503216991661298827085253874986715640259357181211
58010874201927188034331795521745775807492011461248648881199348135404704014035530166993144357741120529114678558810613101815745502012-07-06
server> Line 2:
886815238763772553042134491194026071233240053486
server> Line 3:
859487358508352233872181578144151531399185251849
server> Your certificate is valid.
Press 'Enter' to exit.

client> Connection closed.
```